Algorithme agent logiciel TAKUZU

**Création classe « Agent » correspondant à chaque cellule :**

* Définition des valeur initiales :
  + Def Init (Col, Lig, Grille):
    - Colonne
    - Ligne
    - grille du takuzu
* Définition des conditions :
  + Def Valide (valeur):
    - Si valeur = 0:
      * Alors **i** = 1
    - Si valeur = 1:
      * Alors **i** = 0
    - Si dans la ligne, pour la valeur d'entrée, les cases avec cette valeur correspondent à la moitié des cases de la ligne.
      * Retourner **i**
    - Si dans la colonne, pour la valeur d'entrée, les cases avec cette valeur correspondent à la moitié des cases de la colonne.
      * Retourner **i**
    - Si les 2 cases supérieures sont identiques à la valeur en entrée.
      * Si oui retourner **i**
    - Si les 2 cases inférieur sont identiques à la valeur en entrée.
      * Si oui retourner **i**
    - Si les 2 cases de gauche sont identiques à la valeur en entrée.
      * Si oui retourner **i**
    - Si les 2 cases de droite sont identiques à la valeur en entrée.
      * Si oui retourner **i**
    - Si la case sup et la case inf sont identiques à la valeur en entrée.
      * Si oui retourner **i**
    - Si la case gauche et la case droite sont identiques à la valeur en entrée.
      * Si oui retourner **i**
    - Sinon retourner “”
* Définition de la fonction pour savoir si insérer 0 ou 1:
  + Def Decide\_Val:
    - Tester la fonction “Def Valide” pour la valeur 0 ou 1.

Retourner la valeur de i

**Création de la classe TAKUZU:**

* Définition de la grille de TAKUZU:
  + Def Init (self, grid):
    - self.grid =
    - self.size =
    - self.agents = [[Agent(row, col, self) for col in range(self.size)] for row in range(self.size)]
* Définition de la résolution du TAKUZU :
  + Def Résolution(self) :
    - Pour chaque case :
      * Test que la cellule est vide
        + Si oui exécute la fonction « Decide\_Val » et place la valeur dans la case
        + Si non ne fait rien

**CODE basé sur l’algorithme :**

import tkinter as tk

import numpy as np

class Agent:

def \_\_init\_\_(self, col, lig, grille):

self.col = col

self.lig = lig

self.grille = grille

def valide(self, valeur):

if valeur == 0:

i = 1

elif valeur == 1:

i = 0

else:

return "" # Valeur non valide

# Vérifie les conditions de validité

if np.sum(self.grille.grid[self.lig] == valeur) >= self.grille.size // 2:

return i

if np.sum(self.grille.grid[:, self.col] == valeur) >= self.grille.size // 2:

return i

# Vérifications des cases adjacentes

if self.lig >= 1 and self.grille.grid[self.lig - 1, self.col] == valeur: # Case supérieure

return i

if self.lig < self.grille.size - 1 and self.grille.grid[self.lig + 1, self.col] == valeur: # Case inférieure

return i

if self.col >= 1 and self.grille.grid[self.lig, self.col - 1] == valeur: # Case gauche

return i

if self.col < self.grille.size - 1 and self.grille.grid[self.lig, self.col + 1] == valeur: # Case droite

return i

if self.lig >= 1 and self.lig < self.grille.size - 1 and self.grille.grid[self.lig - 1, self.col] == valeur and self.grille.grid[self.lig + 1, self.col] == valeur: # Case sup et inf

return i

if self.col >= 1 and self.col < self.grille.size - 1 and self.grille.grid[self.lig, self.col - 1] == valeur and self.grille.grid[self.lig, self.col + 1] == valeur: # Case gauche et droite

return i

return "" # Aucune condition remplie

def decide\_val(self):

for valeur in [0, 1]:

result = self.valide(valeur)

if result != "":

return valeur

return -1 # Aucune valeur valide

class Takuzu:

def \_\_init\_\_(self, grid):

self.grid = np.array(grid)

self.size = self.grid.shape[0]

self.agents = [[Agent(col, lig, self) for col in range(self.size)] for lig in range(self.size)]

def resolution(self):

for lig in range(self.size):

for col in range(self.size):

if self.grid[lig, col] == -1: # Cellule vide

valeur = self.agents[lig][col].decide\_val()

if valeur != -1:

self.grid[lig, col] = valeur

if not self.resolution(): # Appel récursif pour continuer la résolution

self.grid[lig, col] = -1 # Revenir en arrière

return False # Aucune valeur ne convient

return True # Puzzle résolu

def display(self):

return self.grid

class TakuzuApp:

def \_\_init\_\_(self, master):

self.master = master

self.master.title("Takuzu Solver")

# Initialiser la grille

self.initial\_grid = [

[-1, 0, 1, -1, -1, 1, 0, -1],

[1, -1, -1, 0, 0, -1, -1, 1],

[0, -1, 1, 1, -1, 0, 1, -1],

[-1, 0, -1, -1, 1, 1, -1, 0],

[-1, -1, 0, 1, -1, -1, 0, 1],

[0, 1, -1, -1, 0, 1, -1, -1],

[1, -1, 0, -1, 1, 0, 1, -1],

[-1, -1, 1, 0, -1, 0, -1, 1]

]

self.takuzu = Takuzu(self.initial\_grid)

self.buttons = []

# Créer l'interface

for lig in range(self.takuzu.size):

row\_buttons = []

for col in range(self.takuzu.size):

btn = tk.Button(master, text='', width=5, height=2, command=lambda lig=lig, col=col: self.toggle\_cell(lig, col))

btn.grid(row=lig, column=col)

row\_buttons.append(btn)

self.buttons.append(row\_buttons)

self.solve\_button = tk.Button(master, text='Résoudre', command=self.solve)

self.solve\_button.grid(row=self.takuzu.size, columnspan=self.takuzu.size)

self.update\_buttons()

def toggle\_cell(self, lig, col):

if self.takuzu.grid[lig, col] == -1:

self.takuzu.grid[lig, col] = 1

elif self.takuzu.grid[lig, col] == 1:

self.takuzu.grid[lig, col] = 0

else:

self.takuzu.grid[lig, col] = -1

self.update\_buttons()

def update\_buttons(self):

for lig in range(self.takuzu.size):

for col in range(self.takuzu.size):

value = self.takuzu.grid[lig, col]

if value == -1:

self.buttons[lig][col].config(text='', bg='SystemButtonFace')

elif value == 0:

self.buttons[lig][col].config(text='0', bg='lightblue')

else:

self.buttons[lig][col].config(text='1', bg='lightgreen')

def solve(self):

if self.takuzu.resolution():

self.update\_buttons()

else:

print("Aucune solution trouvée.")

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

app = TakuzuApp(root)

root.mainloop()